Trust and Connection with Cryptographic and Blockchain Technology - **DRAFT**

How Blockchain and Cryptographic Technology can be used to Personalise Experiences and Enhance Customer Connection for Art Dealers and Luxury Brands

Patrick Mockridge

Aberystwyth, Wales

Abstract

Blockchain and cryptographic technology offers artists and luxury brands the potential to connect with consumers directly and personally in new ways that can radically change the relationship for those who grasp the opportunity.

This paper aims to separate the reality from the hype in this new emerging market and offer a powerful, scalable and simple solution for those interested in building high quality, refined digital relationships between artists, luxury brands and consumers.

I. Technology Background

Blockchains are a decentralised transaction ledger secured mathematically via the use of encryption keys. The private key is known only to the individual or organisation corresponding to the public key or address [1] to which assets are ascribed on the public ledger which is visible to all. The unique problem that Bitcoin [2] solved when it was launched in 2008 was to create a financial settlement mechanism without a need for a central authority.

As Bitcoin grew, developers began to take advantage of a rudimentary coding language within the Bitcoin blockchain in order to extend its functionality. Colored Coins [3], for example, were created to represent the transfer of assets on the Bitcoin blockchain that were separate from Bitcoin itself.

As the number of different rudimentary secondary applications on the Bitcoin blockchain expanded, Ethereum [4] was founded in November 2013. The founder Vitalik Buterin recognised that the potential functionality of blockchains extended far beyond financial transactions and sought to create a new blockchain with a Turing Complete [5] coding language. The conceptual intention was to replicate the success of Netscape's invention of Javascript [6]: which transformed the internet browser from a hyperlink service to a universal application service and laid the foundation for Web 2.0.

While Ethereum has since proven to be wildly popular, subsequent flaws and limitations have been unearthed since its launch in July 2015:

- The DAO Bug

While the risks of Javascript's Turing Completeness were limited to the extra man hours required to fix bugs, bugs in deployed Solidity code can result in decentralised applications (DApps) being drained of funds. The DAO was intended to become the world's first Decentralised Autonomous Organisation, but a bug in the code design to govern the DAO resulted in Ethereum being split, or forked, between Ethereum and 'Ethereum Classic', and a hacker draining the DAO DApp of its Ether deposits.

- Quality Assurance Burden

The fundamental problem uncovered by the DAO Bug [7] was that the Ethereum Virtual Machine (EVM) language, Solidity, is not of the correct construction for the purpose of a blockchain virtual machine. Because Solidity executes synchronously but the blockchain itself executes asynchronously, those coding DApps cannot be sure of the order in which the functions within an EVM DApp will be carried out in production.

This makes Ethereum's Turing Completeness a burden, because while it is capable of performing any computational task, from inspection of the code it is hard to define the possibility space of all outcomes under all circumstances.

This results in a massive quality assurance burden for Ethereum DApps: which is typically a minimum of 40 hours per line of code. At typical engineering rates this amounts to $2000-4000 per line of code on top of the typical iteration cycle. This severely limits the complexity of what Ethereum is able to do in reality.

- Not Much Different from Bitcoin in Reality

The unique value proposition of Ethereum turns out to be a double-edged sword in real world production. Turing Completeness dangles the carrot of being able to do *anything* but the risk of anything outweighs the reward in all but a few cases. Due to commercial risks in production and the associated quality assurance burden, Ethereum DApps are limited to all but a marginal incremental improvement over Bitcoin.

The fundamental technical reasons for are complex and beyond the remit of this paper, but were first explained at Ethereum Devcon 1 in 2015 [8].

Factom [9] was founded in 2014. It extents the functionality of Bitcoin via the two simple core solutions:

- Federated Servers [10]
- Merkle Trees [11]

These two simple solutions together allow for a scalable notarisation and authentication service, rooted in the same immutable cryptographic mathematics that underpin Bitcoin and Ethereum. Factom's Merkle Trees extend the functionality of the Bitcoin blockchain without the congestion and high transaction fees associated with direct blockchain transactions, meaning that anyone can take advantage of the power of cryptographic technology and blockchains for any use case.

II. Blockchain Hype vs Reality

While Ethereum token standards such as ERC-20 (fungible tokens or FTs), ERC-721 (non-fungible tokens or NFTs) and ERC-1155 (a token factory producing both FTs and NFTs) have achieved strong popularity in the five years since the launch of Ethereum, these tokens aren't meaningfully different from what was already possible with Colored Coins on the Bitcoin network prior to Ethereum's launch.

Ethereum NFTs are particularly being touted as a means for asset authentication, without further engineering they are unsuitable for this purpose from an insurable engineering asset perspective for the following reasons:

- Ethereum token metadata is not stored on the Ethereum blockchain itself, only a serial number is stored on chain. Ethereum NFTs alone do not account for how the data of the asset in question is stored and can only point to it.
- Ethereum is not currently scalable, only being able to process roughly 10 transactions per second globally.

In order to address the scalability problem forEthereum tokens, Enjin are developing the E-Finity [14] network. This second layer solution will allow for frictionless scaling trading of assets within the Enjin Ecosystem without interacting directly with the Ethereum blockchain.

For assets where the quality assurance and provenance are key to the value (such as with professional competence, second hand goods or organic food), for public safety and the environment (such as in architecture or engineering), or for building strong personalised relationships between brands and customers (such as with luxury and prestige brands), Ethereum NFTs alone cannot fulfill the conceptual design remit of making and representing an insurable claim about people, goods and services.

III. The Proposed Solution

By combining a three tier-ed solution of combined public storage on Factom federated servers and off chain storage using Git [15], which both use Merkle Trees to prove data integrity, with a simple Ethereum NFT, digital assets can meet any conceivable quality assurance burden (up to and including nuclear power plants, automobiles and skyscrapers) and any conceivable level of refinement and detail of private data in a way that is infinitely scalable.

This system was designed to meet the highest possible quality assurance burden of real world Major Accident Hazard facilities and is therefore currently the best in class conceptual solution.

IV. Git Merkle Tree + Blockchain Diagrams

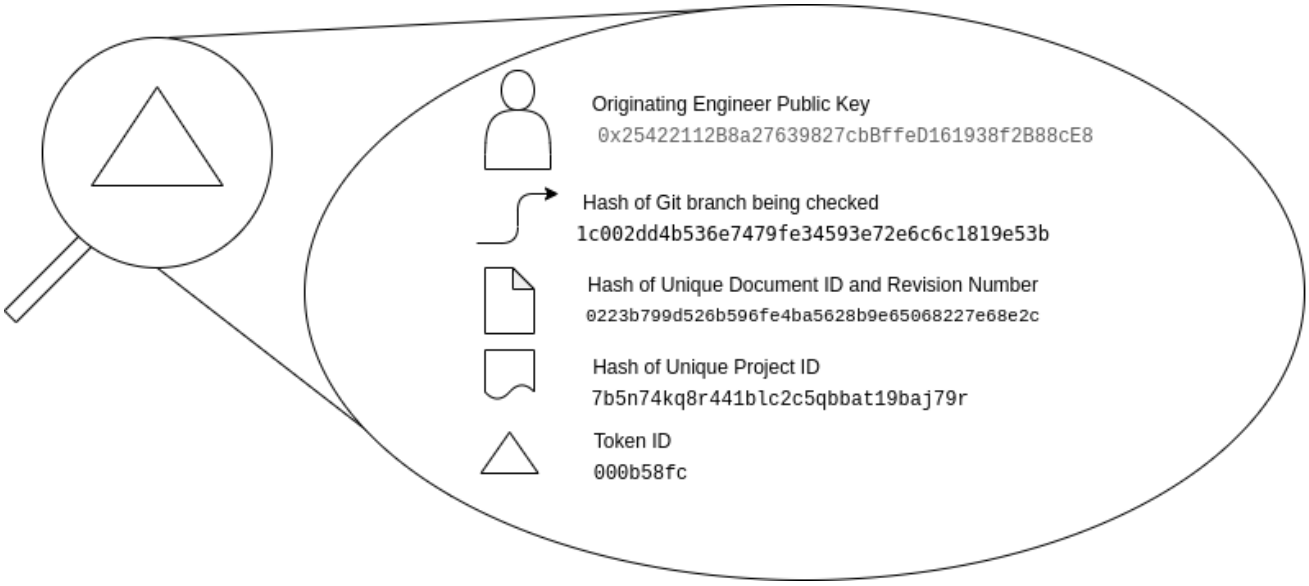

Figure 4.1: The engineering checking cycle begins with an Originating Engineer creating a document under the supervision of a Checking Engineer both of whom working under the direction of a Project Manager. When the Originating Engineer has originated a document according to required specifications to the best of his knowledge and competence he signs it and passes it on to the Checking Engineer. Using Git + Blockchain this creates the first token in the 'Ring of Authentication' Git branch which contains the above properties.

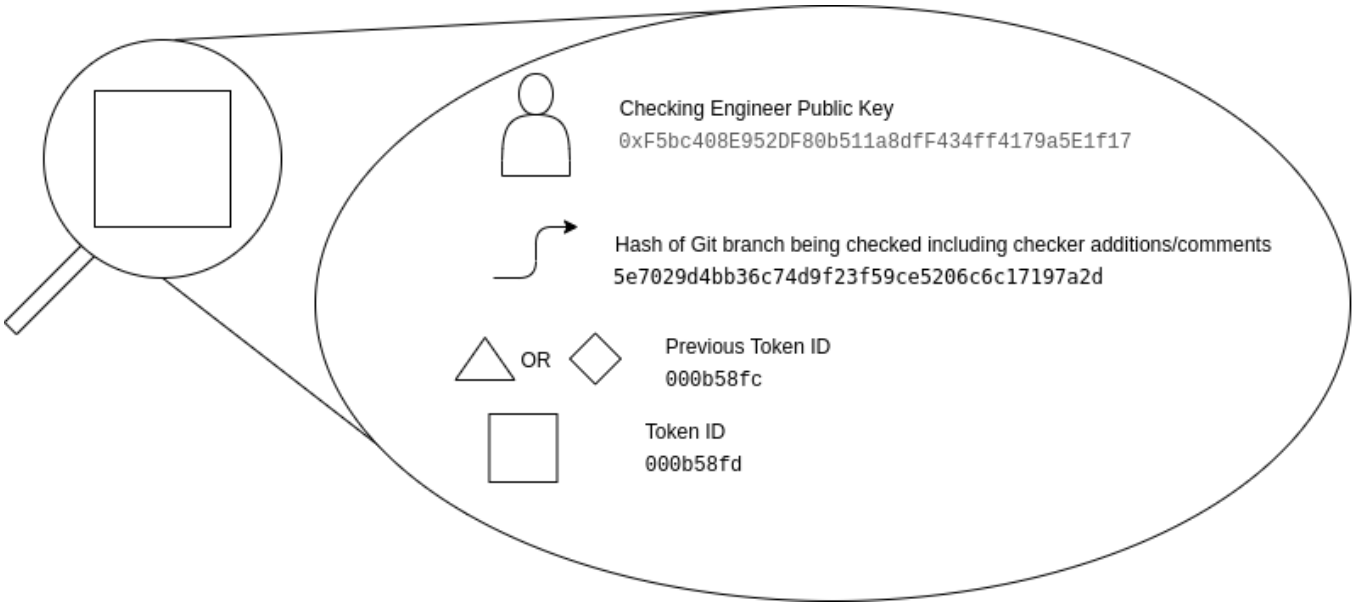

Figure 4.2: When the document has been originated and signed by the Originating Engineer the engineering checking cycle continues from the Originating Engineer to the Checking Engineer, who will be of higher competence and greater experience. The role of the checking engineer will be twofold: firstly he/she will correct any mistakes in the originated document, secondly he/she will add any further information and details outside of the originating engineer's range of expected competence. The Checking Engineer will mark up the originated document before signing it and passing is back to the Originating Engineer for revision. In this way the value of the Checking Engineer's competence and experience is maximised. Using Git + Blockchain this creates the second token in the 'Ring of Authentication' Git branch which contains the above properties.

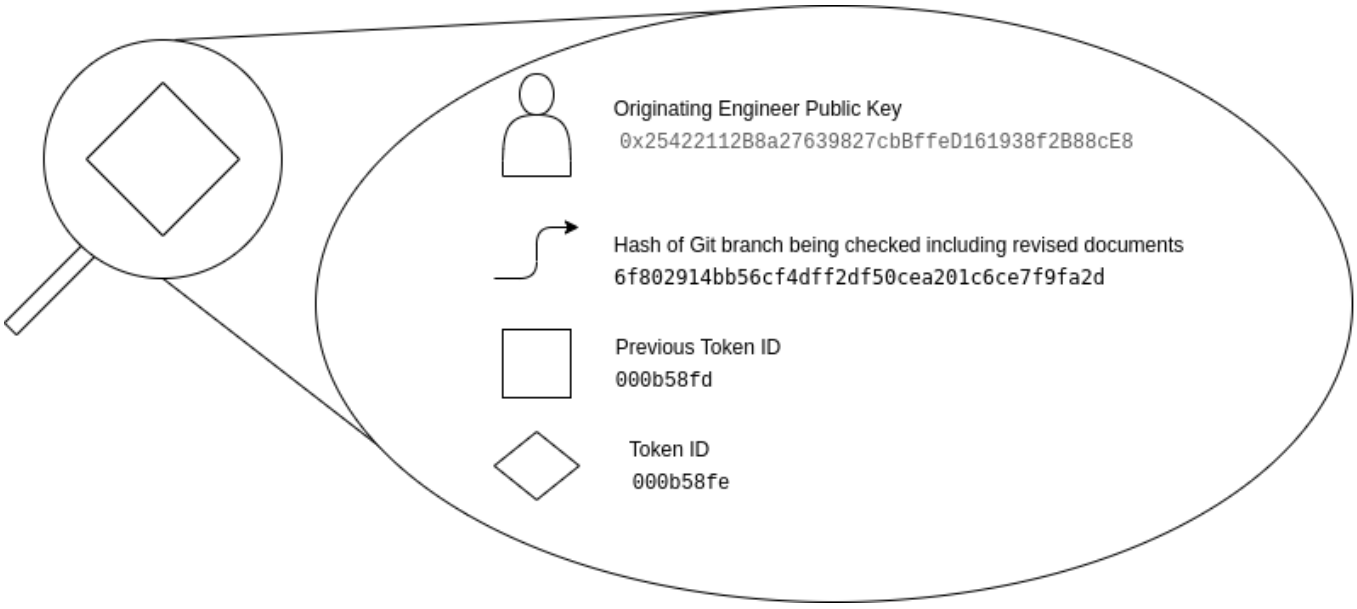

Figure 4.3: When the document has been checked and signed by the Checking Engineer the engineering authentication cycle continues back at the desk of the Originating Engineer who will revise the document by incorporating the comments and corrections of the Checking Engineer. If the Originating and Checking Engineer are working well together there should be only one revision, but the document can be checked and revised any number of times until the checking engineer is satisfied with the result. Using Git + Blockchain this creates the third token in the 'Ring of Authentication' Git branch which contains the above properties.

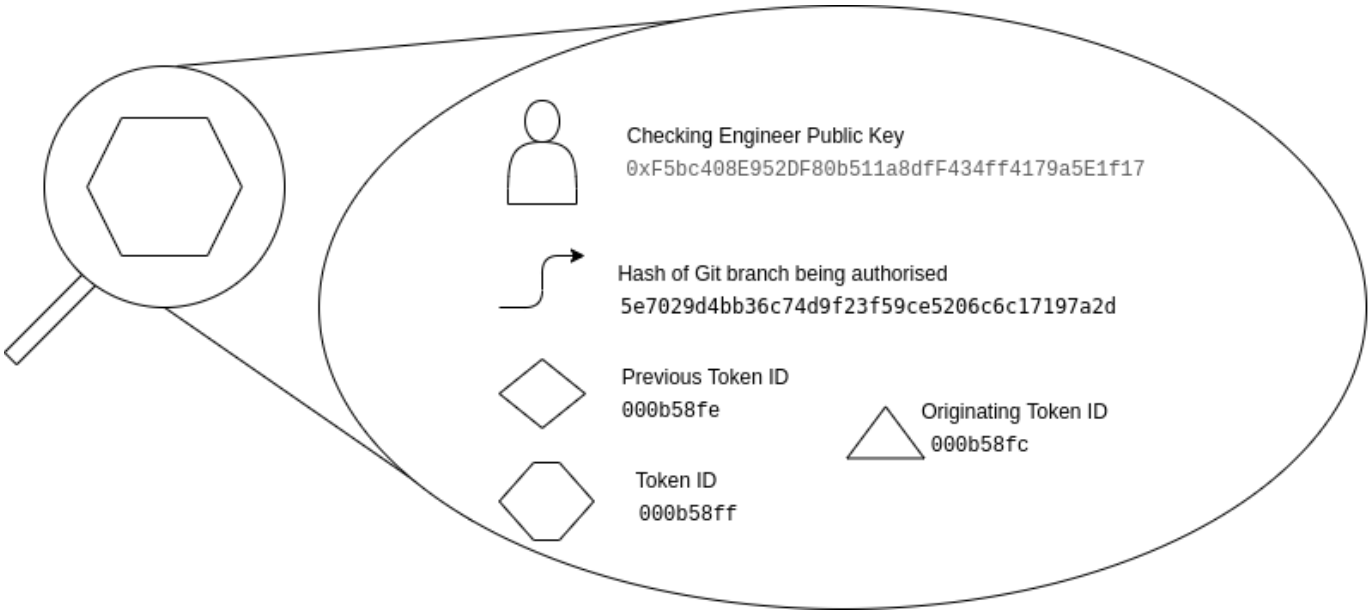

Figure 4.4: When the document has been revised by the Originating Engineer to the satisfaction of the Checking Engineer the Checking Engineer will authorise the document to be integrated into the broader project. Using Git + Blockchain this creates the fourth and final token in the 'Ring of Authentication' Git branch which contains the above properties.

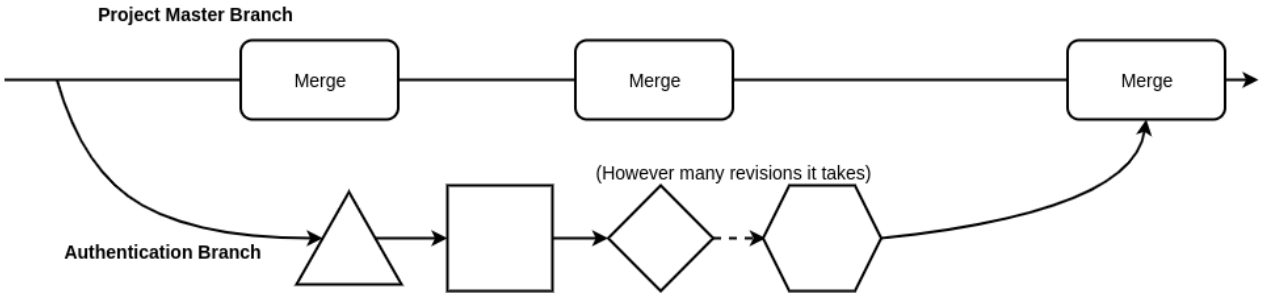

Figure 4.5: The tokenization of the engineering checking cycle creating the Rings of Authentication on the Development or Authentication branch of the project Git repository works as above. The system marries perfectly with Git's existing branching functionality and Merkle Tree data integrity framework.

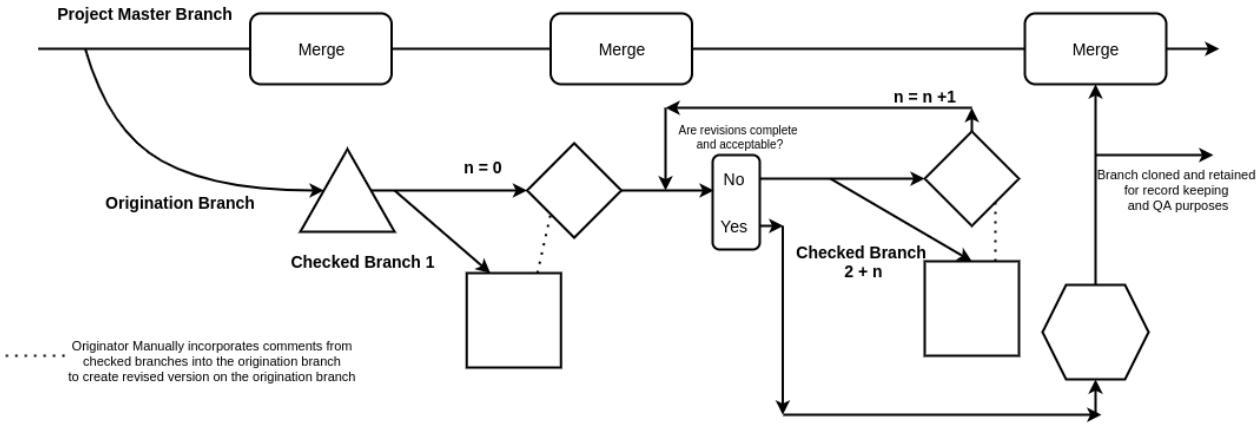
Project Master Branch
Merge
Merge
Merge
n = n +1
Are revisions complete and acceptable?
n = 0
No
Yes
Origination Branch
Checked Branch 1
Checked Branch 2 + n
Branch cloned and retained for record keeping and QA purposes
Originator Manually incorporates comments from checked branches into the origination branch to create revised version on the origination branch

Figure 4.6: Conceptual branching diagram demonstrating how the Checking and Originating Engineers collaborate according to the typical requirements of engineering documentation management. While it appears complex the only major difference between this and the use of Git in software source code development is that both the Checked revision is kept on a separate branch for quality assurance purposes and that the branch itself is cloned and retained prior to merge into the Project Master branch.

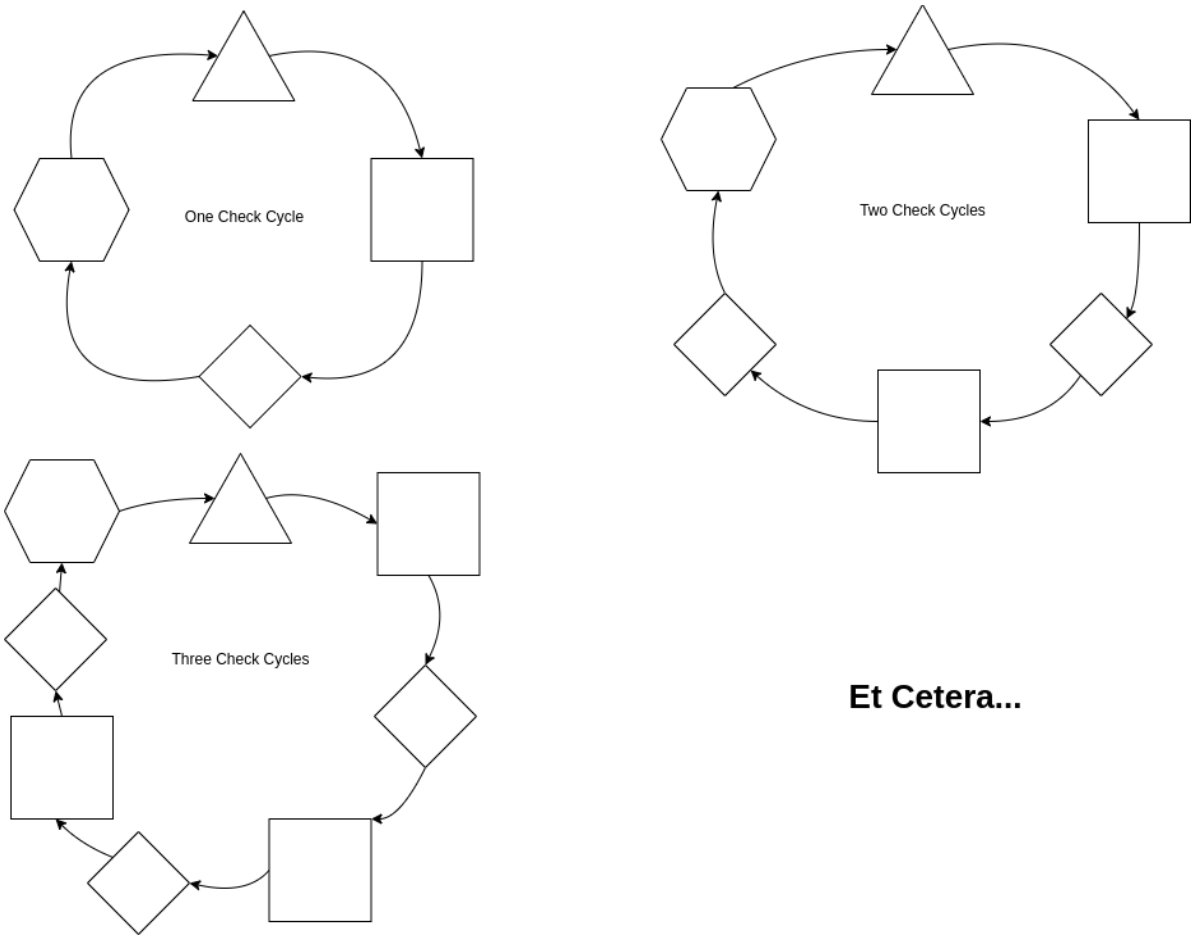

Figure 4.7: When an Originating Engineer and a Checking Engineer are working well together ideally there should never be a requirement for more than two checking cycles, checking cycles can continue indefinitely until the Checking Engineer is prepared to authorise the Ring of Authentication.

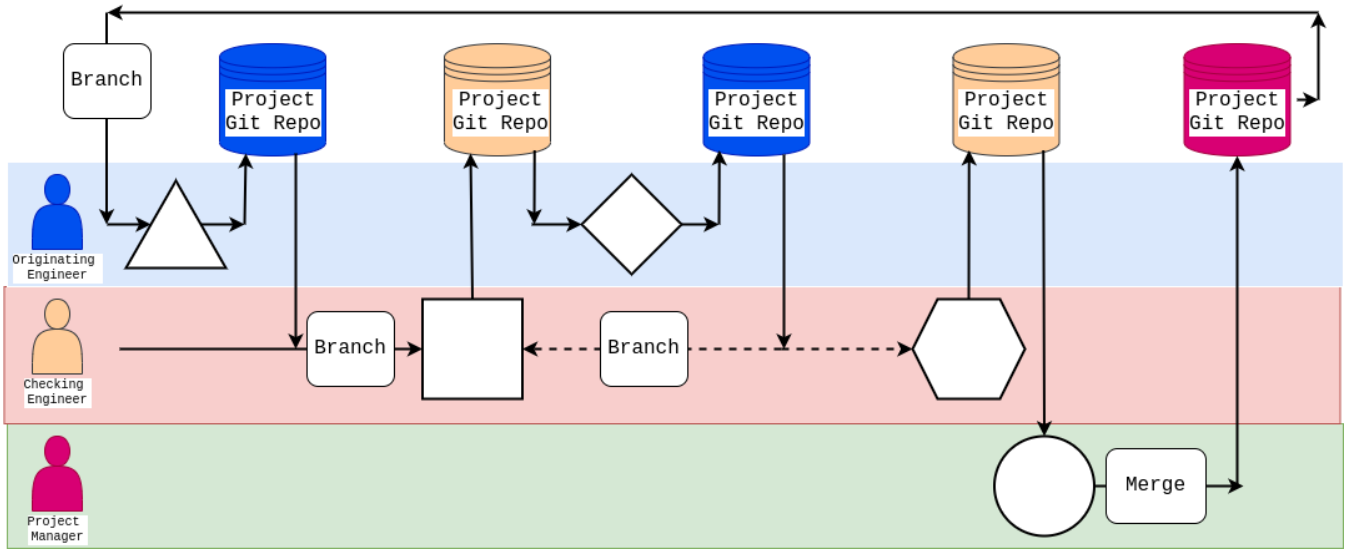

Figure 4.8: The Checking Cycle responsibility diagram. The Project Manager approves the final merge into the Master Branch of the Git Repository.

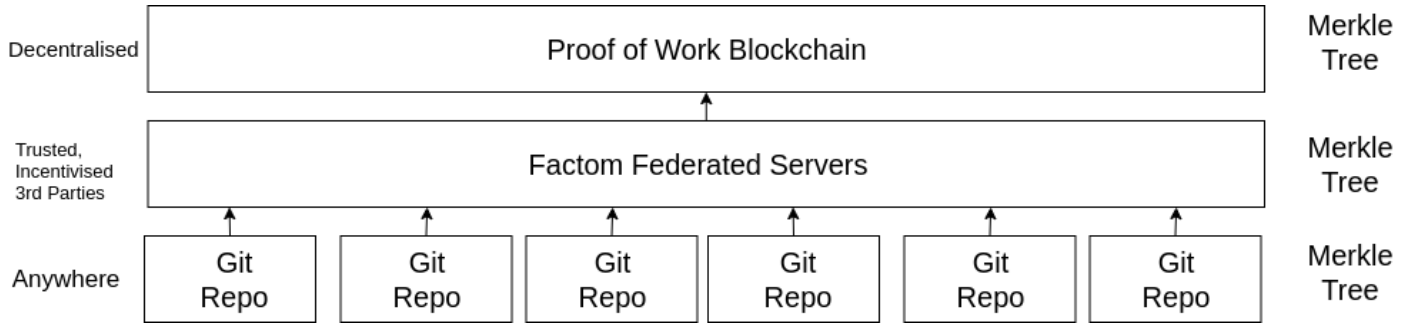

Figure 4.9: While RChain + RSpace offers a highly advanced Git + Blockchain integration with on chain repository storage and object capability access management, the design is robust and simple enough to work well with a basic Merkle Tree federated server solution such as Factom.

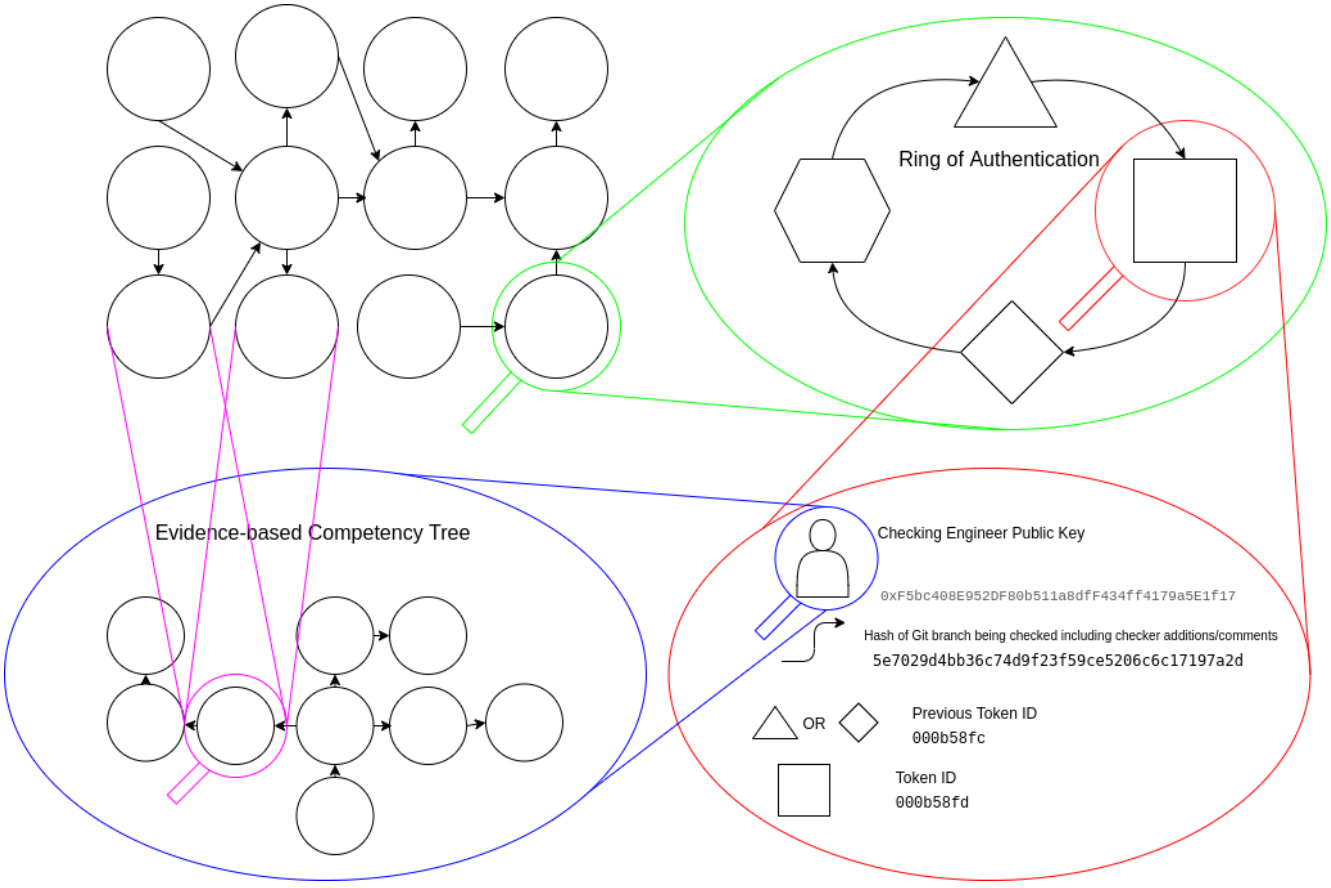

Figure 4.10: In tandem with ENGZIG, explained in section XIV which also uses Git + Blockchain but for professional competency, the process creates a virtuous circle of both refined and transparent quality assurance whereby the provenance of all engineering deliverables can be inspected and traced through a de facto Decentralised Acyclic Graph. This not only greatly improves the insurability of delivered engineering but also strongly discourages the stagflation of engineering signatures and engineered products and strongly encourages engineers to build up their competence, their experience and therefore their reputation in accordance with real world outcomes.

V. Git + Factom

Git is software for tracking changes in any set of files, usually used for coordinating work among engineers. It was initially developed for software engineers developing source code but its functionality can be used for LaTex and word processing files, CSV files, XML files and even CAD files in IFCXML format are beginning to emerge to leverage Git's functionality as part of ISO standards for Building Information Management. Git today can be thought of as a documentation management system as much as a code management system as many more people begin to understand how the software industry is able to collaborate seamlessly using this remote version control system. Git's documentation management goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

Current Established Git Use Cases:

- Software Development
- Video Game Development
- Robotic Development
- Data Science and AI Development

Emerging Use Cases for Git:

- Collaborative Music Composition
- Building Information Management
- Engineering Documentation Management

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with the help of other kernel developers. Since 2005, Junio Hamano has been the central point of authority. As with most other distributed version control systems every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server.

In this way Git is similar to blockchain technology in that every machine contributing to the 'ledger' has a full copy of that ledger. The difference is that Git requires a central competent authority to approve all updates to the 'Main Branch' of the project. But outside of the main branch, work is free to continue in parallel, which enables scalable and robust collaboration. The Linux Kernel, the project for which Git was originally created to serve, accepts commits from over 12 000 software engineers globally and runs on billions of devices. Git is free and open-source software distributed under GNU General Public License Version 2 and is there free to use and free to be extended and expanded to include new functionality.

A major element that Git and all blockchains and DAGS have in common is content addressability. Content addressability means that each piece of data is uniquely identified by the contents of that data, as opposed to a sequential ID number or arbitrary label specified by a user.

For Git the commit ID of each commit is the hash of that commit object. This hash is calculated from all of the data that makes up that commit, including (but not limited to):

- File changes included in the commit
- File names of the committed files
- Committer name
- Commit date and time
- Commit message
- Parent commit ID

The commit object itself is identified by its own content.

A hash is the resulting output from a hash function. Hash functions can map arbitrary blocks of data to fixed-size output chunks, for example, SHA-256 always outputs a 256-bit result. A good hash always has uniform outputs: if you feed it randomly generated inputs, the outputs don't tend to "bunch up" in one part of all combinations of the output possibilities.

For our purpose of cryptographic hash functions, they have a few more requirements:

- The input should not be easy to obtain from the output.
- The output should change drastically in response to even a single change in the input.
- No two inputs should ever map to the same output.

Content-addressable systems are inherently resistant to tampering. To change one piece of data means updating every piece that references it, which in turn means every piece that references that, and so on. When using certain Git commands that rewrite history, such as `git rebase -i` or `git filter-branch`, even one change requires Git to recalculate every commit after the one that was changed, since each commit's identity depends on its parent.

In most of these systems, they create what's called a Merkle tree. A Merkle tree is a tree structure where each node of the tree has an identifier that's the hash of its contents. Git's object database is a Merkle tree and because of the referential nature of Merkle Trees, they are tamper-proof.

The Merkle Tree structure maintains the integrity of both blockchain and DAG technology as well as Git. The beauty and simplicity of Merkle Trees as systems to maintain data integrity is that they can plug into each other by just extending branches ad infinitum. Thus Git's Merkle Tree can extend into any blockchain Merkle Tree trivially giving engineers on projects access to the power of either the Proof of Work of Proof of Stake consensus mechanism for their own work as opposed to blockchain and DAG technology being purely about finance or non-technical consumer focussed products such as collectibles of advertising.

Diagrams for a conceptual model of how this would work are presented in the previous section (Section XII). The model is based upon the typical engineering checking cycle, used by every engineering design house in the world globally today, and replaces ink signatures with the same PGP cryptographic signatures used to sign transactions on both encrypted Git repositories and blockchains themselves. The model assumes that engineers will have full control of their public-private key pairs, ideally stored on a personal hardware wallet. The model also assumes that every engineer is chartered or has obtained a PE certificate such that the work being signed is of commercially insurable quality.

The checking cycle creates rings of authentication for engineering documentation, each of which can be considered to be a block on a decentralised acyclic graph, this DAG of the rings of authentication builds the insurable project documentation structure.

Because Git uses a Merkle Tree structure it can be used effectively with as simple a blockchain integration as Factom [25], which stores chains of data within Merkle Trees on a shared system of decentralised federated servers. The root of the federated server Merkle Trees is pinned to either the Bitcoin or Ethereum blockchains depending upon the economics of the transaction fees at the time.

When projected documentation quality assurance is married with individual competency quality assurance with ENGZIG, a virtuous circle of transparent trust is created.

Of course, using commands such as `git rebase` or `git filter-branch`, would require re-authentication of previous over-written authentications, but because Git is available on a GNU license, between the user interface design and the re-design of Git for the specific purpose of engineering documentation management, this could easily be handled by a good engineering design team.

VI. ENGZIG

ENGZIG is a competency management tool utilising Git and blockchain in order to manage professional reputations in the same way that engineering documentation is managed [17].

In its simplest form it can plug directly into existing solutions such as Factom, but in a more complex form could store Git repositories using RChain RSpace.

In any case, the core premise is that transparent competency profiles for a variety of professions are created by those delivering the work to the highest quality in the real world. Rather than having bureaucratic institutions dictate competency requirements, the governance system will select for a balance between the most popular professionals who are most capable of achieving broad consensus and the most competent professionals most capable of *doing* what the competency map dictates.

The competence maps for any given profession will be public and fully transparent, essentially open source, and professionals will be able to clone them freely in order to create the template their own competence profiles or 'competence trees' which are analogous to 'skill trees' in video games like Final Fantasy XV or Resident Evil 6. These competency trees will be contained within a folder structure that aligns with the competency tree structure. It will use either XML or MD language to take full advantage of Git's ability to determine the difference or 'diff' between revisions of documents. All of this will come with a user interface that makes knowledge of both Git and blockchain technology unnecessary.

The unique value propositions are:

- Competency profiles for common skilled professions are not currently public, transparent and available in one place, making career choices difficult for both university students and minors.
- Because competency profiles are not public, adults themselves must rely upon imperfect information when making career decisions, which can have catastrophic consequences if they go wrong at any time.
- This asymmetry of knowledge gives Power Capital (as described by Nitzan and Bichler) undue influence and control over the lives of the general population and therefore democracy itself.

- Competency maps for some highly skilled engineering professions, such as Chemical Engineering, *aren't even specified* by the institution itself leaving professionals in the lurch between recruiters, HR, companies and Power Capital to determine what constitutes their profession.
- Without peer to peer lateral visibility of competence, it is very difficult for humanity to self-organise organically to build solutions to problems such as climate change.

VII. Ethereum NFT with Git + Blockchain

Using Git + Factom for provenance and quality assurance, when a product is ready for sale an NFT can be created with a single Git hash created at the point that the item has passed all quality assurance checks.

This hash will act a unique identifier and pointer for the item in question and prove mathematically and immutably that the item Ethereum NFT correlates directly to the item in question and its unique chain of authentication which can be checked and validated at every stage via checking the hashes of Git Repository against the authentication rings on Factom.

VIII. RChain Token and DAppy

In terms of token solutions currently in Beta testing RChain Token with the DAppy [19] browser being developed by Raphaël Fabre offers interesting new functionality. While Ethereum Solidity DApps require 40 hours of quality assurance per line of code, RChain Rholang DApps require less than one hour per line, radically changing the economics and iterative timecycles of blockchain development.

RChain itself is a correct by construction, concurrent blockchain developed by Greg Lucius Meredith, inventor of Microsoft BizTalk and global expert in decentralised and distributed computing solutions. Process Calculus and Concurrency are both explicit within RChain's RhoLang coding language, making it perfect for coding decentralised applications which require an appreciation of both in execution. The first 'Mercury' release of RChain will be able to support at least 1000 transactions per second, 100 times that of Ethereum and Bitcoin, with a solid conceptual design basis to later achieve 40 000 transactions per second from the subsequent iteration: 'Venus'.

RChain Token was initially developed by Raphaël Fabre as a clone of the Ethereum ERC-1155 token factory in the Rholang language. While ERC-1155 took months of quality assurance from dozens of developers and contributors, because of Rholang's inherent safety, Raphaël deployed the first RChain Token within three weeks working by himself.

In addition to the faster development time and faster development iteration cycle, RChain Token allows for token metadata, such as pictures, code, pdf and word documents, to be stored directly on the RChain blockchain and therefore to be owned directly via a public-private key pair. RChain Token also supports Object Capability permission patterns allowing users to share ownership of token or groups of tokens and also to build highly complex and refined permissions covering almost any specific pattern for any real world use case.

DAppy is a web browser that interacts directly with the RChain blockchain, from which HTML and media can be served directly. This differentiates it from every other Ethereum DApp today, which serve HTML and media from centralised servers. This has the power to radically change the ownership model of the internet in line with the original Ethereum conceptual vision.

RChain, DAppy and RChain Token are all currently in Beta development.

IX. References

1. Wikipedia: PGP Encryption https://en.wikipedia.org/wiki/Pretty_Good_Privacy
2. Wikipedia: Bitcoin: https://en.wikipedia.org/wiki/Bitcoin
3. Bitcoin Wiki: Colored Coins: https://en.bitcoin.it/wiki/Colored_Coins
4. Wikipedia: Ethereum: https://en.wikipedia.org/wiki/Ethereum
5. Wikipedia: Turing Completeness: https://en.wikipedia.org/wiki/Turing_completeness
6. Wikipedia: Javascript: https://en.wikipedia.org/wiki/JavaScript
7. Medium: The DAO Exploit for Beginners: https://medium.com/@MyPaoG/explaining-the-dao-exploit-for-beginners-in-solidity-80ee84f0d470
8. Youtube: DEVCON1: Monadic Design Patterns for the Blockchain - Lucius Greg Meredith: https://youtu.be/uzahKc_ukfM
9. Factom: https://www.factom.com/
10. Wikipedia: Federated Architecture: https://en.wikipedia.org/wiki/Federated_architecture
11. Wikipedia: Merkle Trees: https://en.wikipedia.org/wiki/Merkle_tree
12. ERC-20 Token Standard: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/
13. ERC-721 Token Standard: https://ethereum.org/en/developers/docs/standards/tokens/erc-721/
14. Enjin E-Finity: https://www.youtube.com/watch?v=C4ex_SLSpZ4
15. Wikipedia: Git: https://en.wikipedia.org/wiki/Git
16. The Distributed Nature of Git and Bitcoin (Initial Commit): https://initialcommit.com/blog/git-bitcoin-distributed-nature
17. ENGZIG: The future of career management: https://vimeo.com/169265783
18. Jonathan Nitzan, Shimshon Bichler, Capital as Power: A Study of Order and Creorder, ISBN-10 9780415496803
19. DAppy: https://dappy.tech/
20. Wikipedia: Object Capabilities: https://en.wikipedia.org/wiki/Object-capability_model